CS380: Introduction to Computer Graphics

Lab Session 1: Hello World 2D

Juil Koo

Spring 2023
KAIST

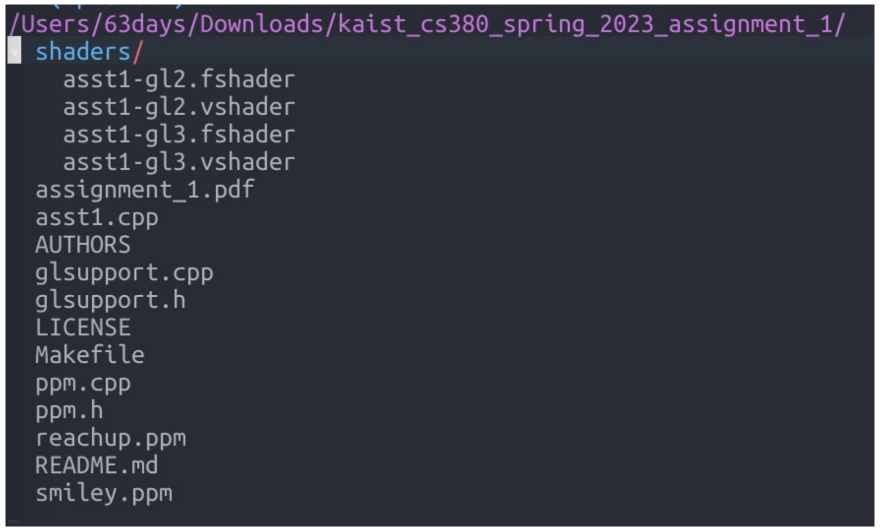
Goals

- Understand a rendering pipeline of OpenGL.
- Practice with shader code.

Assignment Files

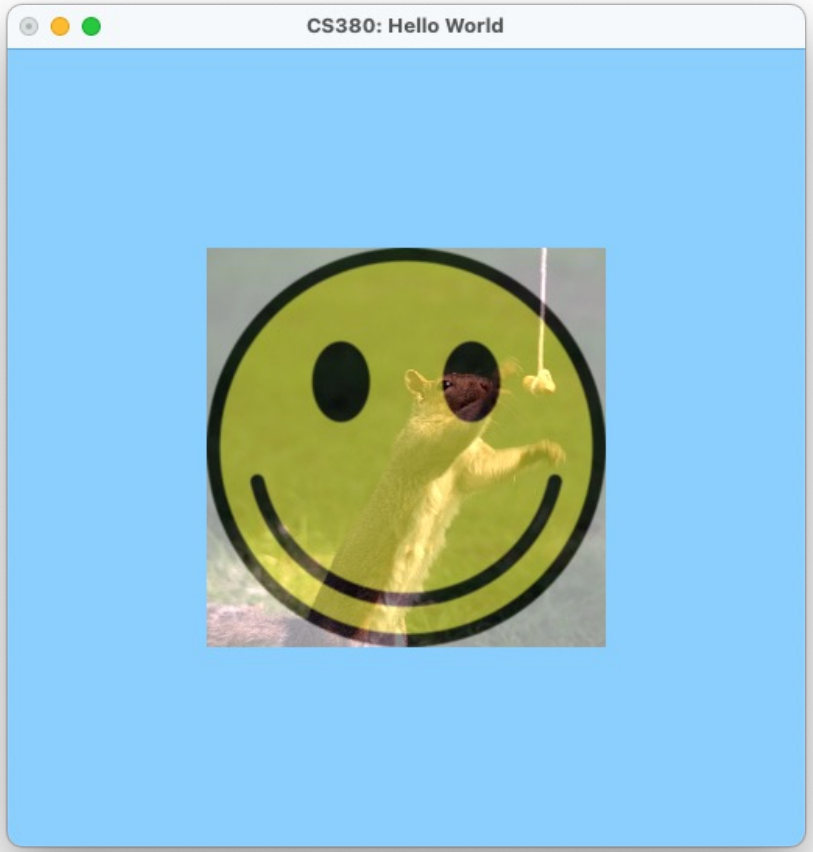
The assignment 1 zip file contains:

- asst1.cpp (main cpp file)
- Shader files (vertex shader and fragment shader)
- Other util .cpp and .h files.
- ppm images.



```
/Users/63days/Downloads/kaist_cs380_spring_2023_assignment_1/
  shaders/
    asst1-gl2.fshader
    asst1-gl2.vshader
    asst1-gl3.fshader
    asst1-gl3.vshader
  assignment_1.pdf
  asst1.cpp
  AUTHORS
  glsupport.cpp
  glsupport.h
  LICENSE
  Makefile
  ppm.cpp
  ppm.h
  reachup.ppm
  README.md
  smiley.ppm
```

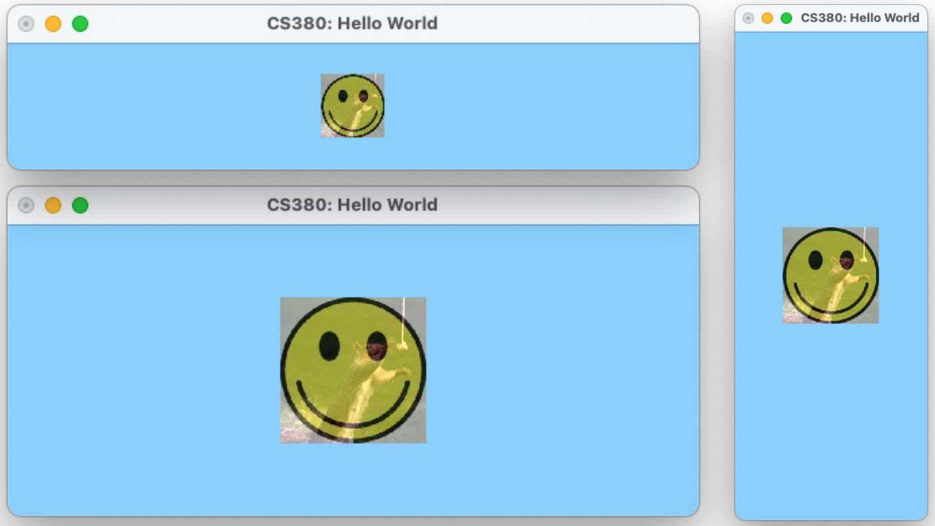
First Screen



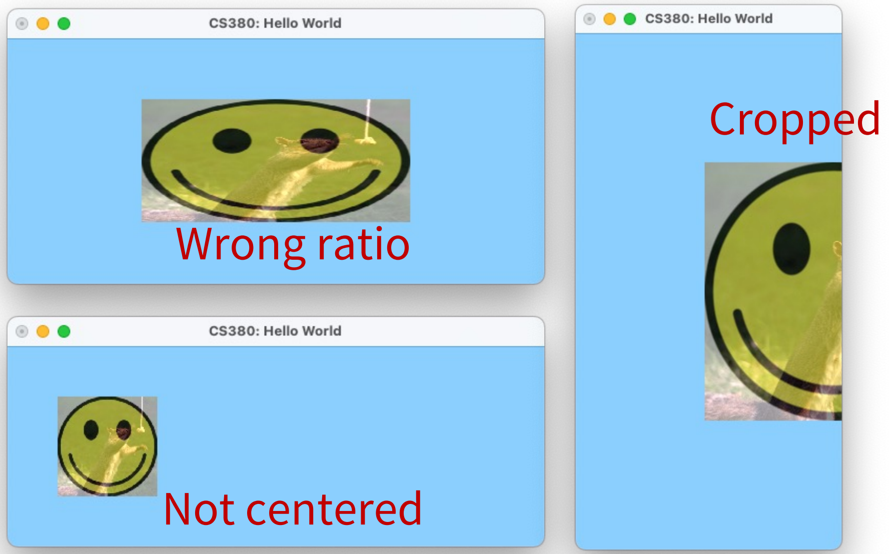
Task

The smile image should place in the center maintaining half size of a shorter direction.

Correct

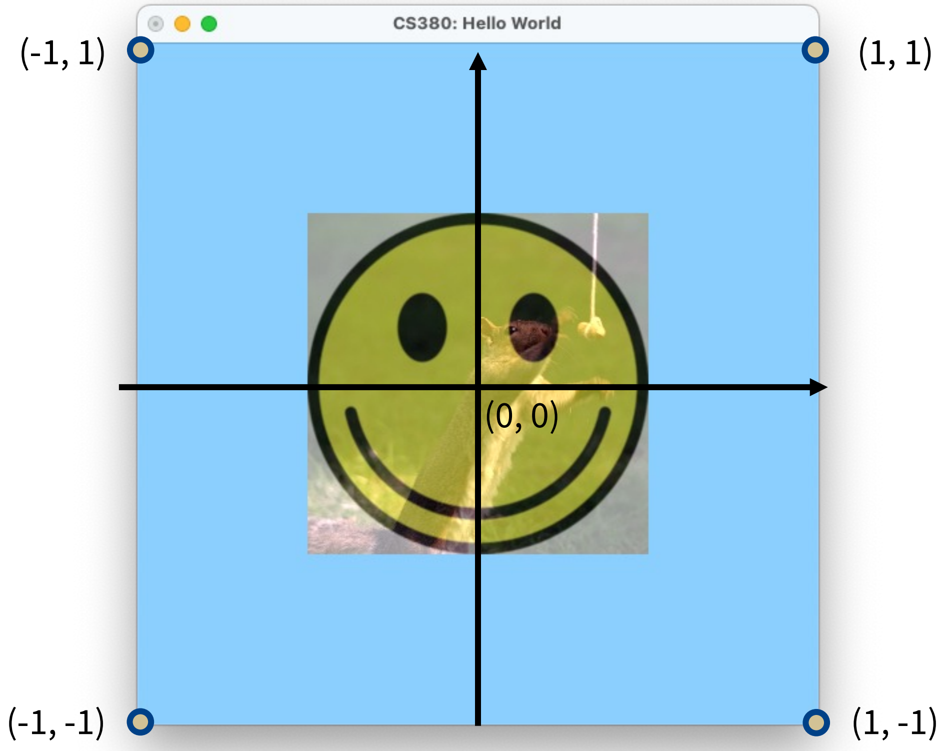


Wrong cases



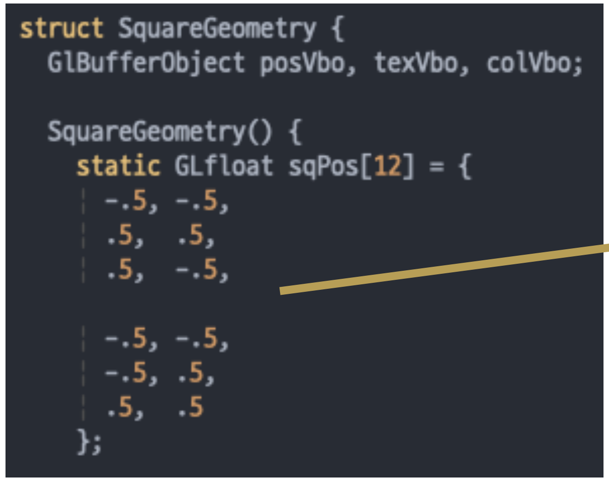
Normalized Viewport Coordinate

Normalized viewport coordinates are within a range of [-1, 1].

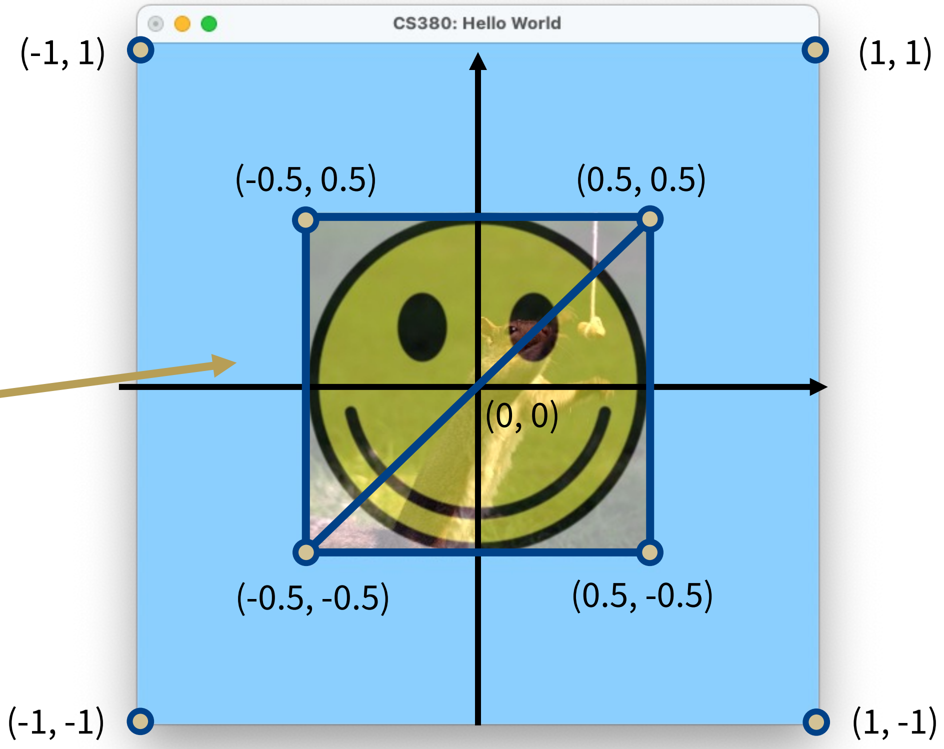


Normalized Viewport Coordinate

The initial coordinates of vertices are defined in `sqPos`.

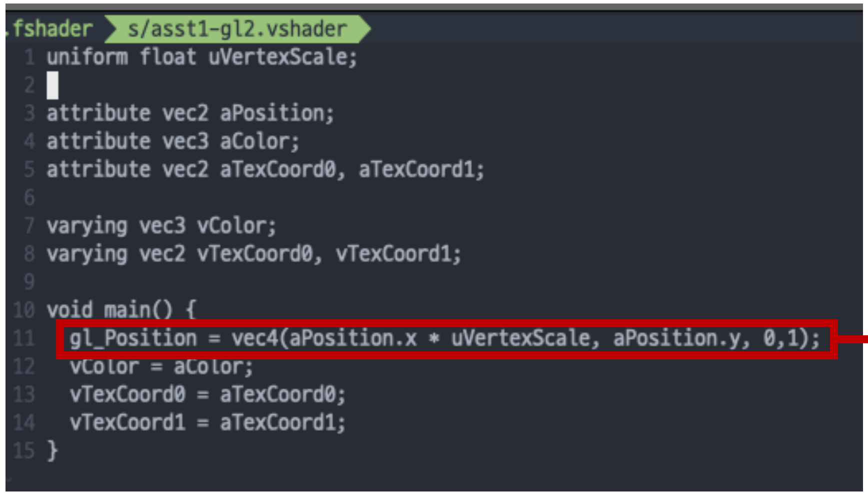


```
struct SquareGeometry {
  GlBufferObject posVbo, texVbo, colVbo;

  SquareGeometry() {
    static GLfloat sqPos[12] = {
      -.5, -.5,
      .5,  .5,
      .5,  -.5,

      -.5, -.5,
      -.5, .5,
      .5,  .5
    };
```



Vertex Shader

- There are three variable types.
  - uniform
  - attribute (= in in gl3)
  - varying (= out in gl3)
- It outputs `gl_Position.`

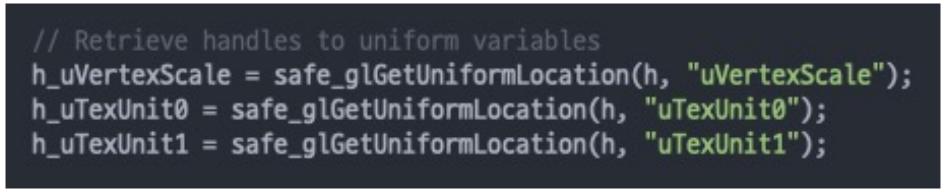


```
uniform float uVertexScale;

attribute vec2 aPosition;
attribute vec3 aColor;
attribute vec2 aTexCoord0, aTexCoord1;

varying vec3 vColor;
varying vec2 vTexCoord0, vTexCoord1;

void main() {
  gl_Position = vec4(aPosition.x * uVertexScale, aPosition.y, 0,1);
  vColor = aColor;
  vTexCoord0 = aTexCoord0;
  vTexCoord1 = aTexCoord1;
}
```

Compute a homogeneous coordinate (x,y,z,w) of current vertex.

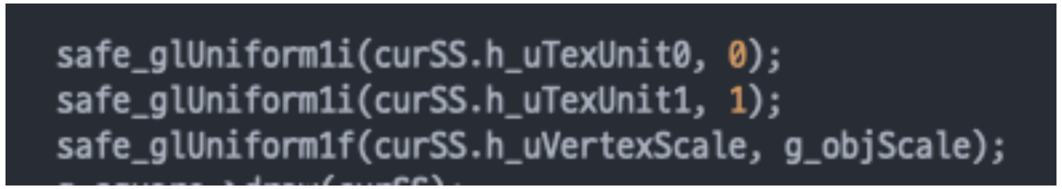
Uniform Variable

- Uniform variables are shared across all vertices.
- Set uniform variables in an application:
  - First get location.



```
// Retrieve handles to uniform variables
h_uVertexScale = safe_glGetUniformLocation(h, "uVertexScale");
h_uTexUnit0 = safe_glGetUniformLocation(h, "uTexUnit0");
h_uTexUnit1 = safe_glGetUniformLocation(h, "uTexUnit1");
```

  - Then set current value.

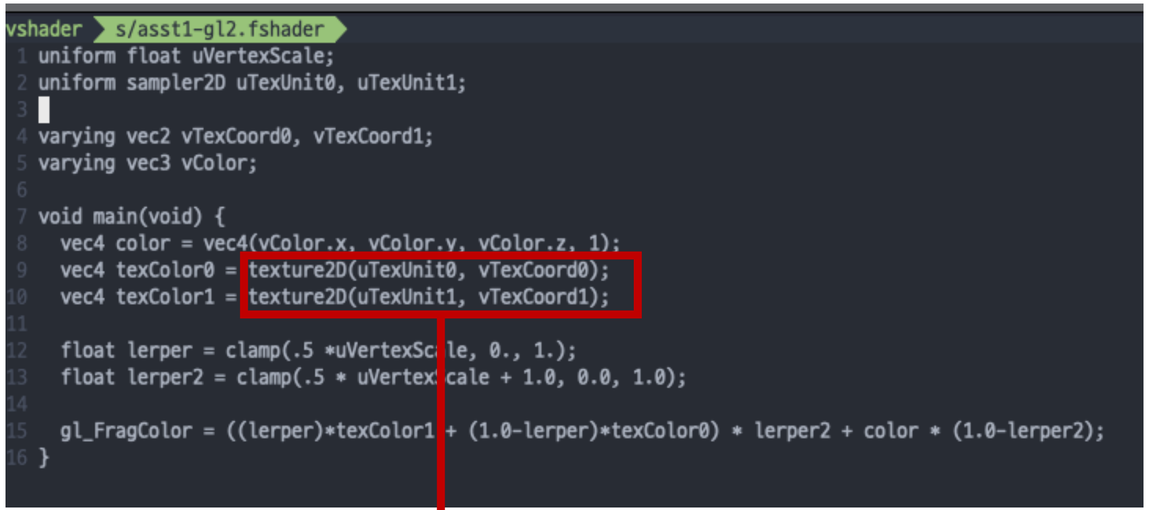


```
safe_glUniform1i(curSS.h_uTexUnit0, 0);
safe_glUniform1i(curSS.h_uTexUnit1, 1);
safe_glUniform1f(curSS.h_uVertexScale, g_objScale);
```

First get the “location” and set the value.

It can pass values to vertex- and fragment-shader.

Fragment Shader

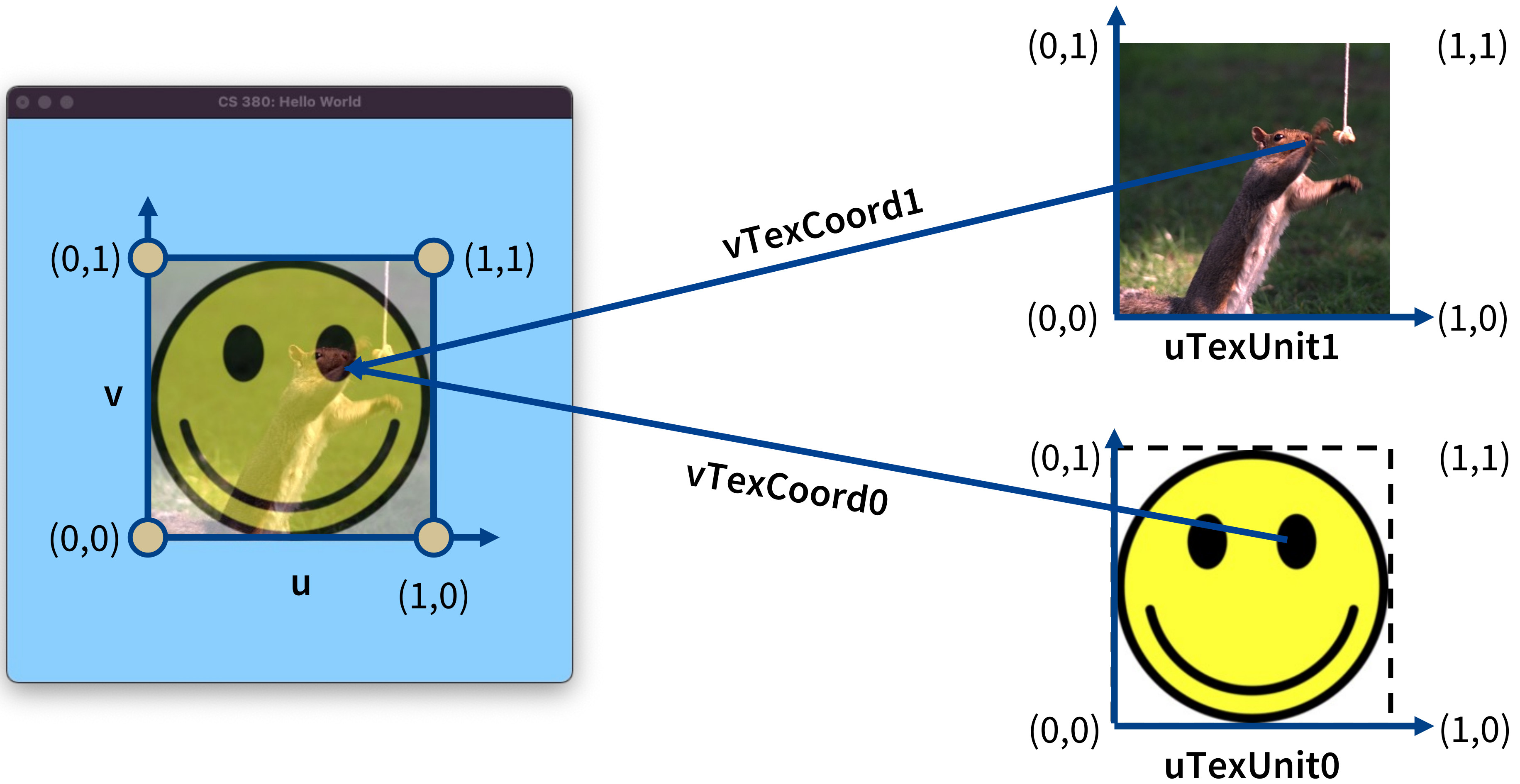


```
uniform float uVertexScale;
uniform sampler2D uTexUnit0, uTexUnit1;

varying vec2 vTexCoord0, vTexCoord1;
varying vec3 vColor;

void main(void) {
  vec4 color = vec4(vColor.x, vColor.y, vColor.z, 1);
  vec4 texColor0 = texture2D(uTexUnit0, vTexCoord0);
  vec4 texColor1 = texture2D(uTexUnit1, vTexCoord1);

  float lerper = clamp(.5 *uVertexScale, 0., 1.);
  float lerper2 = clamp(.5 * uVertexScale + 1.0, 0.0, 1.0);

  gl_FragColor = ((lerper)*texColor1 + (1.0-lerper)*texColor0) * lerper2 + color * (1.0-lerper2);
}
```

texture2D(sampler2D sampler, vec2 coord)
sampler: texture image.
coord: texture coordinates.

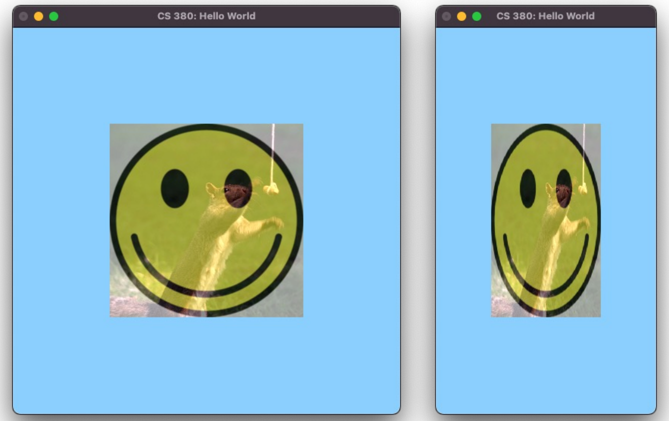
Texture Mapping

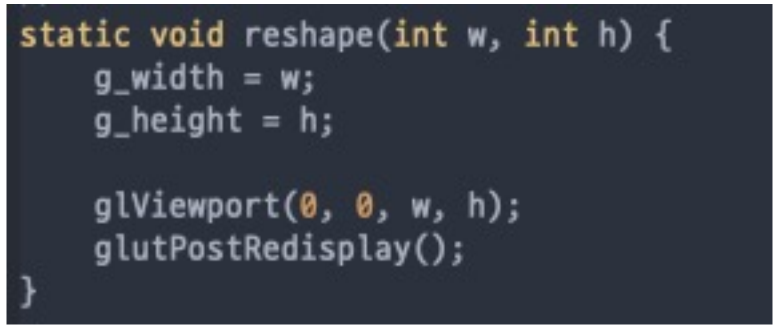
```
vec4 texColor = texture2D(uTexUnit[X], vTexCoord[X])
```



Reshaping

Detect changing window size → Call reshape function





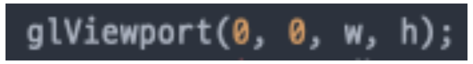
```
static void reshape(int w, int h) {
    g_width = w;
    g_height = h;

    glViewport(0, 0, w, h);
    glutPostRedisplay();
}
```

- glViewport sets the width and height of the window.
- glutPostRedisplay() calls the display callback function.

Constraints for Assignment 1

- Your solution CANNOT change the vertex coordinate.

- Your solution CANNOT modify the call to glViewport.
  - Do not change the following line.



```
glViewport(0, 0, w, h);
```

- You might need to declare and handle additional variables in the vertex shader file.

- You can refer to how uVertexScale in the vertex shader works by clicking a right mouse button.

How to Submit

- Take at least two screenshots, one with a shorter width and the other with a shorter height.

- Compress the files including both the screenshots and your code, and submit a zip file on GradeScope.

- Due date: Mar. 19 (Sun) 23:59 KST.

- If you complete your assignment during the lab session, you can get confirmation from the TA and leave. But, note that your final score will be reassessed based on the submitted code and results. Also, it’s not mandatory to get confirmation.